Tracing Internal Communication in MPI and MPI-I/O

Julian M. Kunkel[1], Yuichi Tsujita[2], Olga Mordvinova[3], Thomas Ludwig[4]

[1] DKRZ, Hamburg
[2] Kinki University, Hiroshima
[3] Ruprecht-Karls-Universität, Heidelberg
[4] Universität Hamburg, c/o DKRZ, Hamburg

PDCAT 2009

DKRZ

Outline

- Introduction
- PIOviz
- Tracing MPI Internals
- Evaluation
  - Collective Communication
  - MPI-I/O
- Summary

Introduction

- Users rely on efficient MPI implementation
- HPC environment is complex
  - Network topology
  - Node hardware
  - Parallel file system
- MPI abstracts from environment
  - Implementations tend to work in multiple environments
  - Might deliver suboptimal performance

Why is Tracing MPI-Internals Useful?

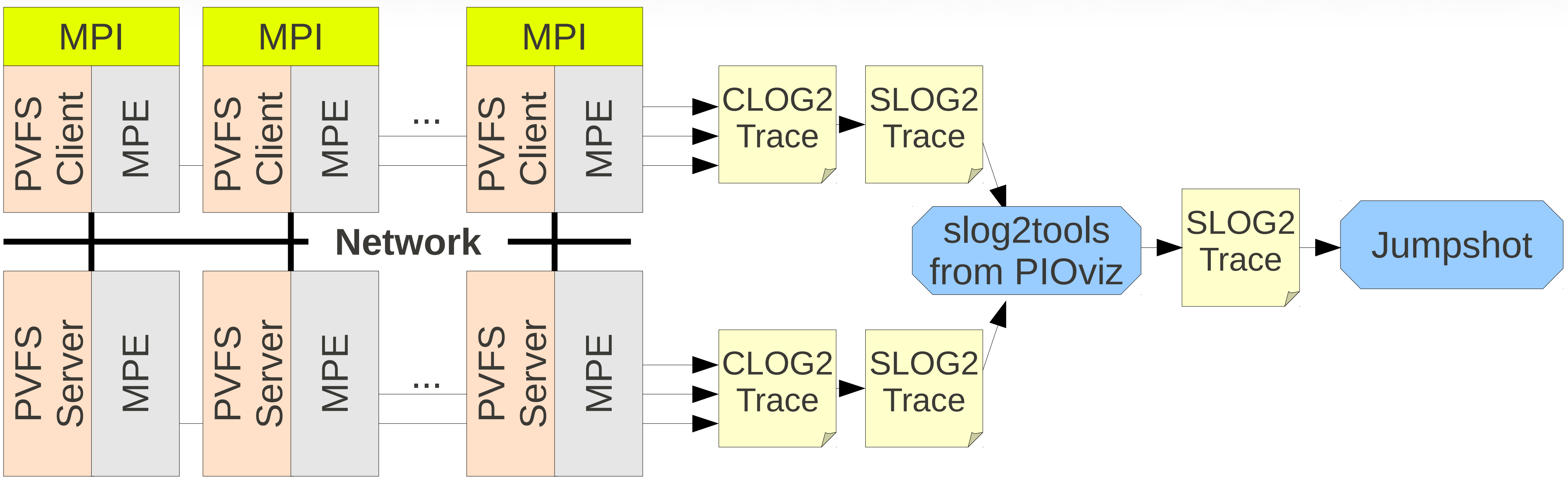
- Users want to
  - Assess MPI performance
  - Optimize MPI
  - Make sure HPC environment is healthy
- Internal processing in MPI depends on application

  => application context is important!
- Understanding processing might improve load balancing

PIOviz

- Tracing environment for
  - MPI applications
  - Server side file system specific information
  - Visualize file system clients and servers together
- Software components:
  - MPICH2
  - PVFS v2
  - Postprocessing scripts
  - Extensions to MPE and Jumpshot

Tracing with PIOviz



- Trace MPI clients with MPE
- Trace PVFS servers with MPE
- Postprocess with slog2tools (excerpt):
  - Merge client and server trace files
  - Correlate client and server activities
- Visualize postprocessed trace with Jumpshot

New in PIOviz

Tracing MPI Internals

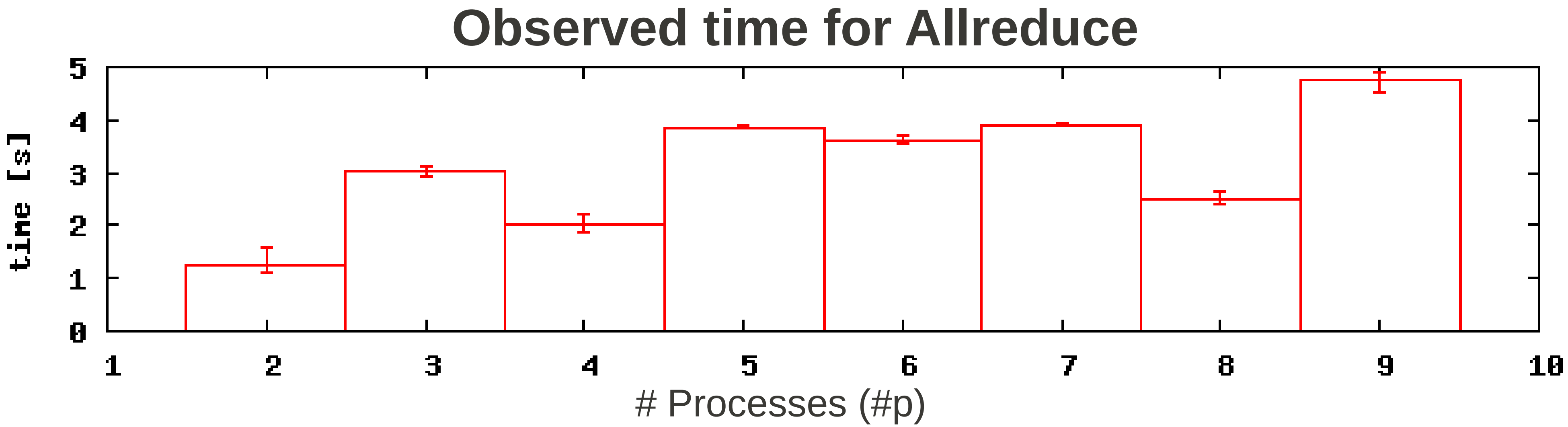
- Modified MPI-I/O calls to use MPI_X calls internally
  - Before PMPI calls were used
- Instrumented PVFS calls inside MPI-I/O layer
- Instrumented internal functions for collective calls
  - Used by collective functions for communication

=> Internal processing of collective functions is traced

Evaluation

Allreduce

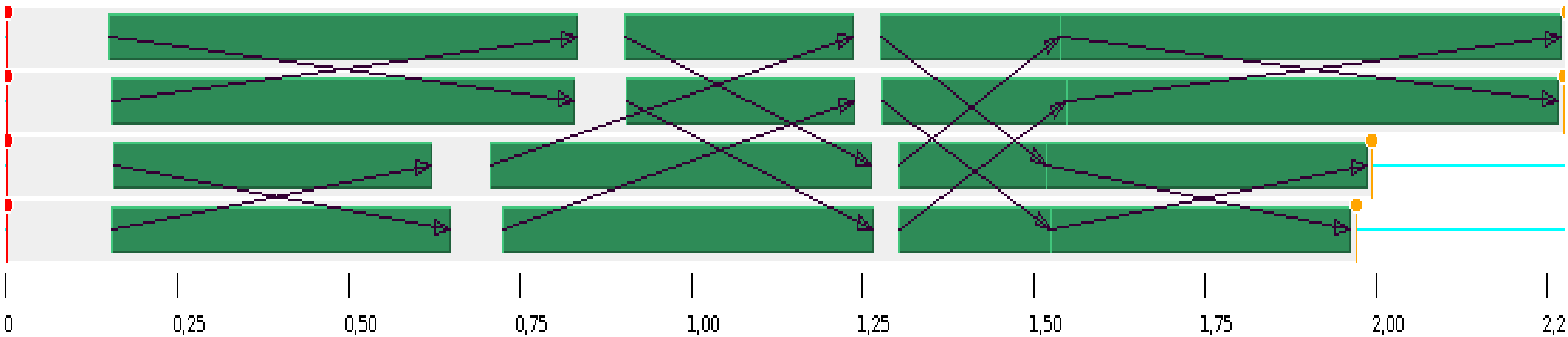
- Experiment
  - Sum 10 million double values (80 Mbyte)
  - 10 times repeated



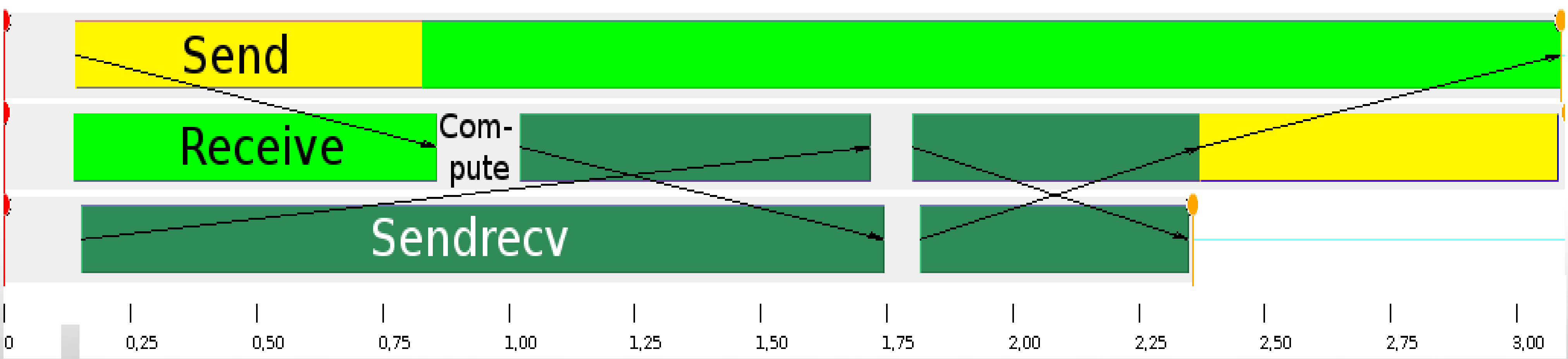
- We expect t(#p) <= t(#p+1)
  - But slower for process # which are not a power of two!

Inside Allreduce

- 4 processes: Binary tree algorithm (all to all)



- 3 processes: First process delays processing



Concluding Allreduce

- Efficient algorithm for allreduce was described
- But not completely implemented!

  => Performance degredation if #p != 2^x

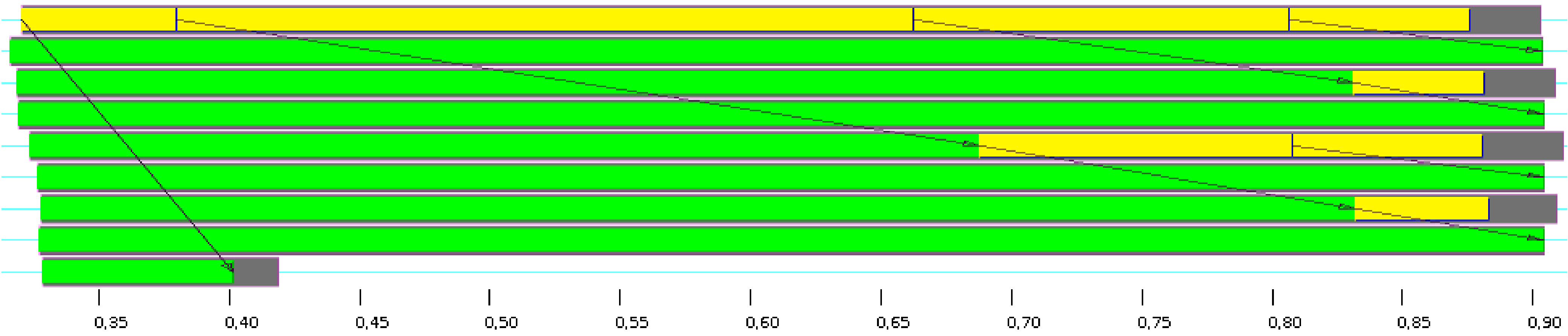
  => As efficient as 4 times the number of processes

- Instead of $t \sim (\lceil \log_2(\#p) \rceil)$ we get $t \sim (\lceil \log_2(\#p) \rceil + 2)$ and load imbalance!

(This is just an approximation)

Inside Scatter

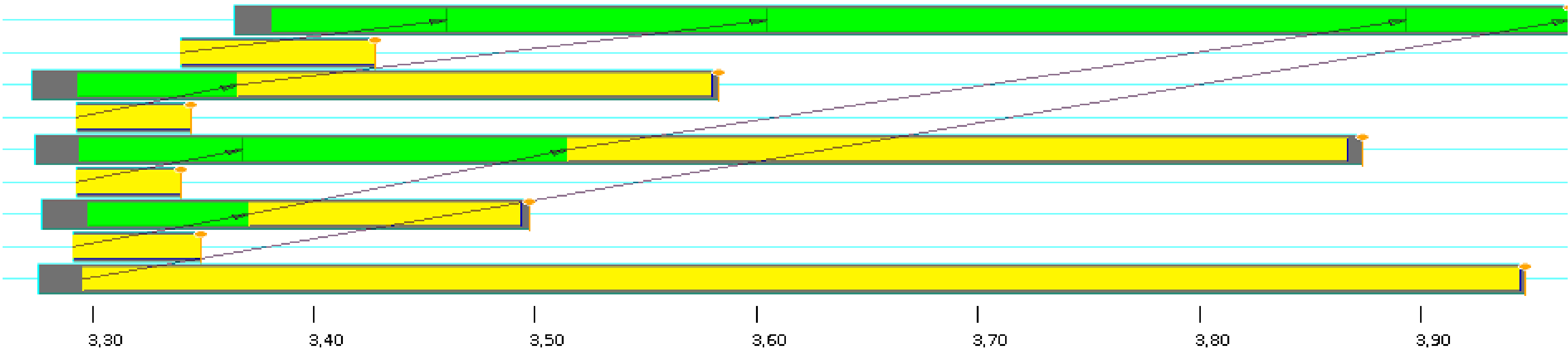
- Scatter 1->9 Clients
- 8 Mbyte of data



- Processes forward data
  - Critical in a switched network topology (except for small msgs)
- All processes (except one) finish at the same time

Inside Gather

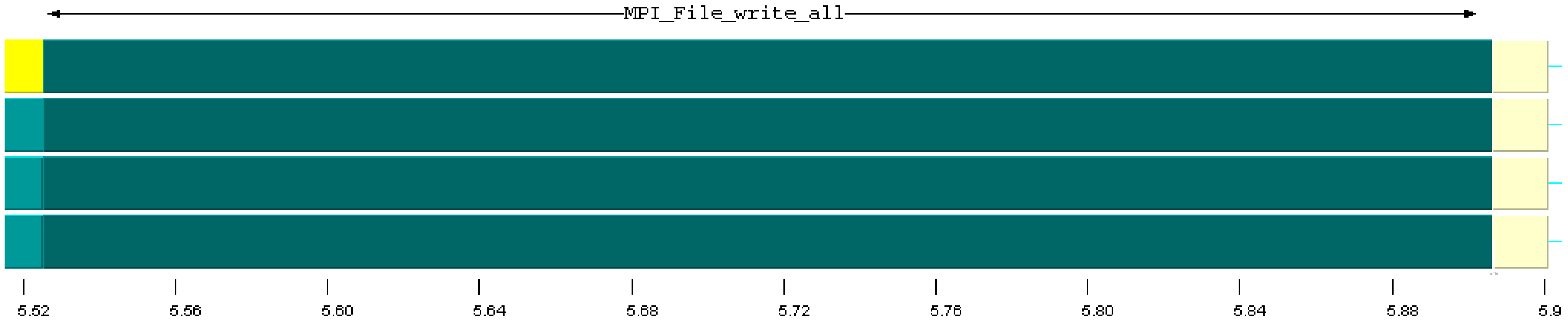
- Gather 9 -> 1 client
- 8 Mbyte of data



- Again forwarding of data
- Load imbalance due to call
  - Nice to put less work on „forwarders“

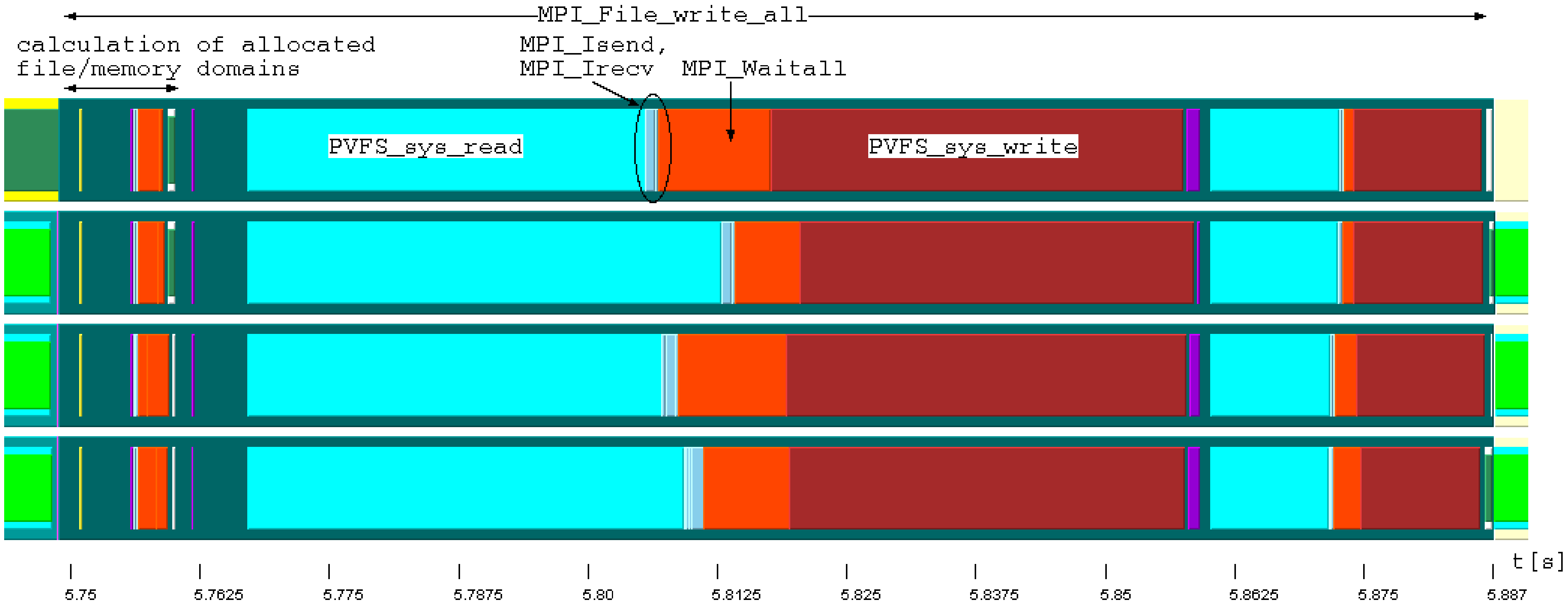
MPI-I/O

Tracing without PIOviz



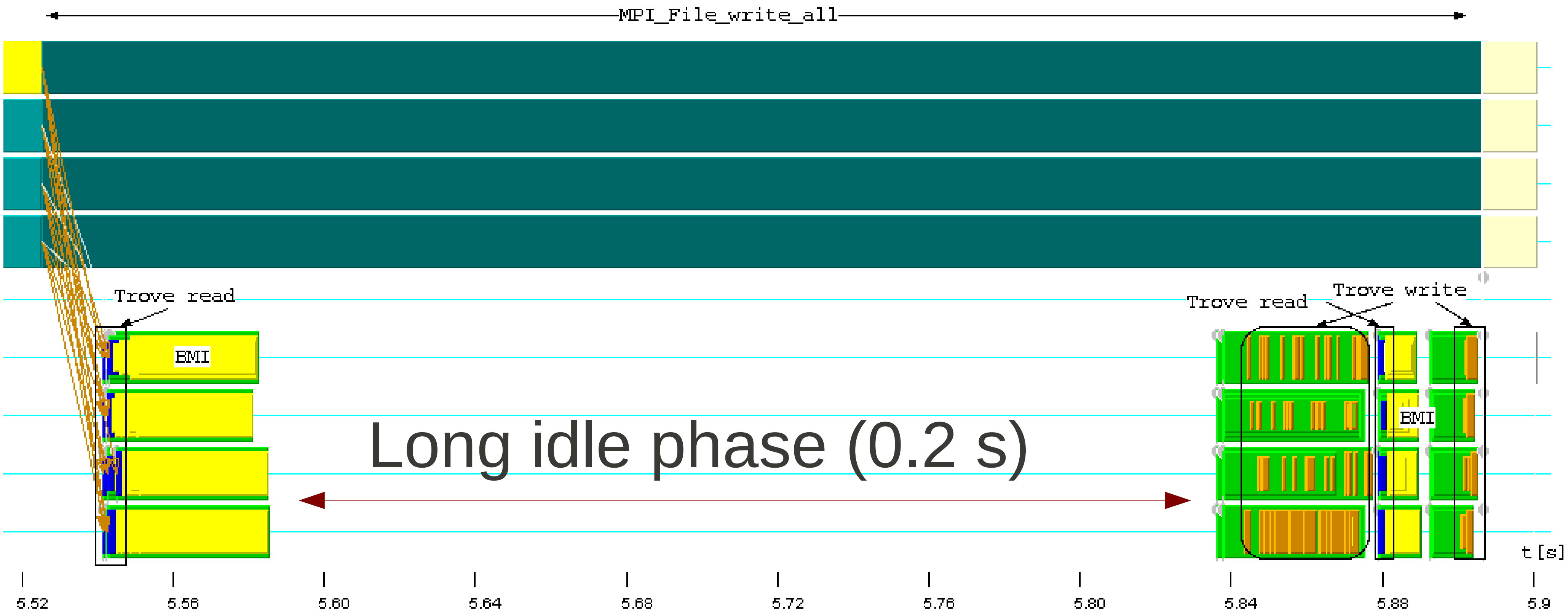
- We cannot assess performance of File_write_all

Tracing Client Internals



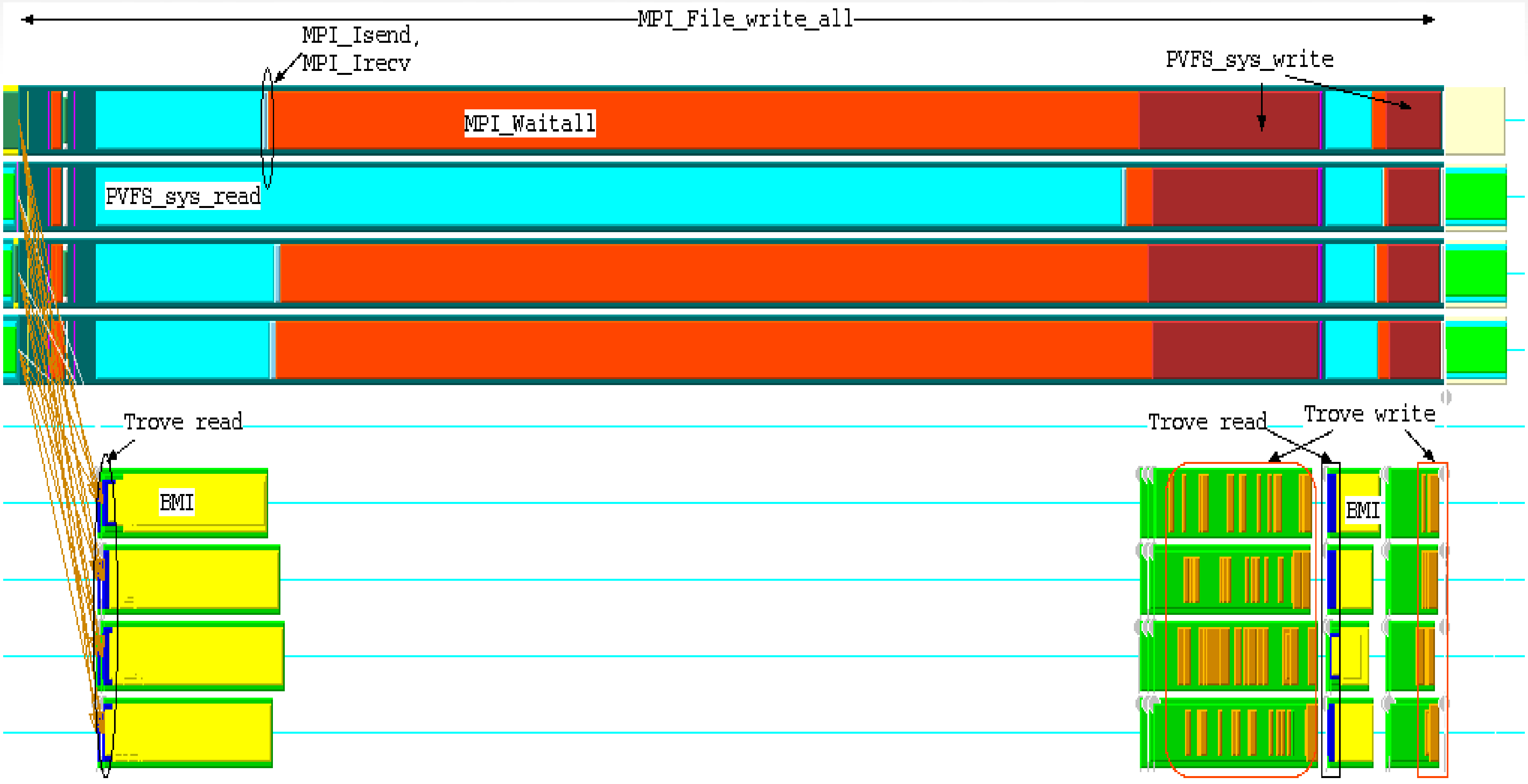
- How performant are the PVFS servers?

PIOviz without Client Internals



- Where is the performance bottleneck?

PIOviz with Client Internals

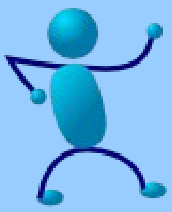


- One client needs a long time to finish PVFS_sys_read

  => Bug inside PVFS Client Library

Conclusions

- Tuning of MPI libraries is important
- We trace application, server and MPI internals
- Revealed suboptimal handling of collective calls
- Combined trace for parallel file system client and server allows
  - to localize bottlenecks
  - to tune internal layers



Thank you for your attention!

かんしゃ